Slices: A Scalable Partitioned for Finite Element Meshes[†]

Hong Q. Ding*   Robert D. Ferraro*

Abstract

A parallel partitioned for partitioning unstructured finite element meshes on distributed memory architectures is developed. The, element based partitioned can handle mixtures of different element types. All algorithms adopted in the partitioned are scalable, including a communication template for unpredictable incoming messages, as shown in actual timing measurements.

1 Introduction

Partitioning a finite element mesh among the processors of a parallel supercomputer sets up the stage for the finite clement analysis problem, The domain partition achieves load balance, preserves proper data locality and reduces communications during the solution of the problem. To handle large meshes and to avoid excessive IO, our partitioned partitions the mesh concurrently on the parallel computer, in contrast, to most misting partitioners which partition mesh on a sequential computer and then download the partitioned mesh to a distributed memory computer.

2 Mesh Decomposition

The element-based partitioning differs from a grid-based partitioning?, (see [1] and references therein) in that an element belongs to a processor in its entirety. This implies that the processor subdomain boundaries go along the edges, instead of cut across the edges in a grid-based partitioning. To facilitate the element based calculations, wc further require that all the nodal points (nodes) of the element must 1 ›e on this processor too. Thus a node on processor subdomain boundaries is replicated on all processors which share it, although onc processor *owns* it during the later solution phase of the linear system which involves vectors defined on the nodes.

3 Basic Strategy

Associating each clement to its center of mass (centroid), the resulting collection of centroids are partitioned via a grid-based partitioned which uses a recursive inertial bisection algorithm, i.e., in each recursive step, the remaining mcsh subdomain is cut into two across its current longest extension.

Once element centroids arc partitioned, elements and nodes migrate from their current processors to the correct processors. In the process, subdomain boundary nodes arc identified and replicated. A fast stochastic algorithm is implemented to balance the owned nodes iteratively to nearly perfect load-balance. With the scalable algorithms implimcntcd, this part takes about the same processing time as the partitioning of centroids.

[†] Work funded by the NASA HPCC ESS Project.

* Jet]'repulsion Laboratory, California Institute of Technology, Pasadena, CA 91109.

4 Template for Unpredictable Incoming Messages .

A data request protocol frequently occurs in the migration of elements and nodes. For example, the already partitioned centroids request that the element structures migrate to the processor where the centroid structures are. The requesting processor know whom to send requests, but the receiving processor does not know how many messages it should expect and how long each message is? This is a problem of unpredictable incoming messages.

We designed a scalable (no worse than the logarithm of number of processors) communication template to resolve this problem as the following; (a) sort data requests on sending processor according to the destinations, (b) call two global communication routines global_ sum () and global_ maximum() so that each receiving processor knows how many messages it should expect and the maximum message length; (c) **make** correct number of calls to receive the requests with the maximum message length it expects.

Once data requests are received, each processor send the requested data back to the requesting processors. Elements and nodes migration arc implemented using this communication template. Minor modifications to the template codes are made to handle the complications duc to the variable number of nodes each finite element could have and due to the variable number of processors that a node is shared.

5 Further Refining the Mesh

The elements/nodes distribution resulted from the partitioner allows one to use a sequential refiner to refine the local mesh independently, without reference to any information not locally available.' An algorithm exists to match the newly created nodes along the subdomain boundary, thus connecting local meshes into a global one. A doptive refinements and multi-level solution methods to the resulting linear equations could be easily added.

6 Connection to a Sparse Solvers Package

The sparse linear equations arising from finite element analysis based the partitioned mesh can be solved by an existing sparse matrix parallel solvers package that wc have written (see the article by the same authors.) The user dots the physics part ,i.e. calculates matrix elements of the sparse stiffness matrix, and calls the solvers to construct the matrix and solve the linear system by either a preconditioned bi-conjugate gradient, method, or a two-stage Cholcksy factorization method, or a hybrid method combining both.

7 Scaling Characters

Wc measured the scaling behavior on increasing the problem size in proportion" to the number of processors on Intel Delta. On 4-processor, the partitioner takes 0.21 scc to partition the 512 clement problem (each element is S-node hexigon). The 4096-element problem on 32-processor takes 0.51 sec. while the 32768-element problem on 25 G-processor takes 0.93 sec. If wc take 4-processor as the minimum processor size where a partition algorithm make sense and normalize all timing accordingly, a logarithmic scaling is clearly followed for this scaled size problem: $T(P)$ /T(4) $= 0.8 log_2$ (P/4), for P processors.

References

[1] S. T. Barnard, A. Pothen, and H. D. Simon, A Spectral Algorithm *for Envelope Reduction of Sparse Matrices, in* Proceedings of Supercomputing 93, IEEE Computer Society Press, Los Alamitos, CA. p.493.